What is claimed is:

1. A floating thread structure for programming threads within a multithreaded application program, the floating thread structure minimizing thread switching overheads and memory usage during execution of the multithreaded application program, the floating thread structure comprising:

   a. a main level function, the main level function being the basic execution level function for the thread, the main level function making subsequent calls to all other functions; and

   b. sub-functions within the main level function, the sub-functions following a function calling convention, the sub-functions comprising:

      i. non-preemptive functions, the non-preemptive functions being normal functions that do not stall the thread at any stage of execution, the non-preemptive functions being called from the main level function or any sub-function called by the main level function, the non-preemptive functions being capable of making calls to other non-preemptive functions only;

      ii. preemptive functions, the preemptive functions being functions that may cause a thread to temporarily stall further execution, the preemptive functions being called from the main level function only, the preemptive functions being capable of making calls to other non-preemptive functions only; and

      iii. other standard program constructs relevant to the thread.

2. The floating thread structure as recited in claim 1 wherein each function definition is preceded by a keyword declaration, the keyword declaration distinguishing amongst main level function, preemptive functions and non-preemptive functions.

3. A method for programming a multithreaded application program using floating threads, the floating threads being threads that do not require saving context information during thread switching, the method comprising:

a. writing a main level function for each floating thread, the main level function being the basic execution level function for the thread, the main level function making subsequent calls to all other functions; and

b. writing sub-functions of the main level function in accordance with a function calling convention, each sub-function being called by the main level function, the writing of sub-functions comprising:

i. writing non-preemptive functions, the non-preemptive functions being normal functions that do not stall the thread at any stage of execution, the non-preemptive functions being called from the main level function or any sub-function called by the main level function, the non-preemptive functions being capable of making calls to other non-preemptive functions only;

ii. writing preemptive functions, the preemptive functions being functions that may cause a thread to temporarily stall further execution, the preemptive functions being called from the main level function only, the preemptive functions being capable of making calls to other non-preemptive functions only; and

iii. writing other program constructs relevant to the thread.

4. The method as recited in claim 3 wherein the preemptive function is written in a manner that the function restarts from its beginning if it preempts the thread.

5. The method as recited in claim 4 wherein writing the preemptive function further comprises:

a. ascertaining the state of the thread, the thread state being ascertained by checking the value of the condition code for the thread, the condition code being a special field associated with the thread, the value stored in the condition code field being indicative of the thread state;

b. performing application specific logic in accordance with the preemptive function;

c. ascertaining whether the function needs to preempt the thread, the ascertaining being based upon the condition code and other application specific logic; and

d. preempting the thread in case the function needs to preempt the thread, the preemption being done after setting the condition code to an appropriate value and performing other application specific logic.

6. The method as recited in claim 3 further comprising declaring a keyword before each function definition, the keyword declaration distinguishing amongst main level function, preemptive functions and non-preemptive functions.

7. A method for minimizing thread switching overheads and number of execution stacks used by threads in multithreaded processing, the method comprising:

a. compiling the threads in accordance with a function calling format, the function calling format ensuring that the thread need not save large context information during thread switching; and

b. executing the compiled application code.

8. The method as recited in claim 7 wherein compiling the threads in accordance with the function calling format comprises:

a. compiling a main level function, the main level function being the basic execution level function of the thread, the main level function making all subsequent calls to other functions;

b. compiling non-preemptive functions, non-preemptive functions being normal functions that do not stall the thread at any stage of execution, the non-preemptive functions being called from the main level function or any sub-function called by the main level function, the non-preemptive functions being capable of making calls to other non-preemptive functions only;

c. compiling preemptive functions, the preemptive functions being those functions that may cause a thread to temporarily stall and yield control to the operating

system, the preemptive functions being called from the main level function only, the preemptive functions being capable of making calls to other non-preemptive functions; and

d. compiling other program constructs.

9. The method as recited in claim 8 wherein compiling the main level function comprises:

a. providing instructions for calling a preemptive function comprising:

i. providing instructions for storing and updating operations in response to the preemptive function call, the operations being performed so that the necessary context information pertaining to the thread is not lost if the thread is preempted during the preemptive function call;

ii. providing instructions for enabling the preemptive function to distinguish between the first and subsequent calls to it, the subsequent calls being made in case the preemptive function preempts the thread;

iii. providing instructions for facilitating switching back of the thread, the switching back being necessary in case the preemptive function call preempts the thread, the facilitating being done in such a manner that the thread is restarted from the beginning of the preemptive function during a subsequent call; and

iv. providing instructions for calling the preemptive function with appropriate parameters; and

b. providing instructions for other program constructs and non-preemptive function calls.

10. The method as recited in claim 9 wherein providing instructions for storing and updating operations in response to a preemptive function call comprises:

a. providing instructions for storing all live local and temporary variables on local memory, the live variables being those variables that have been defined and assigned a value prior to the preemptive function call, the values so defined being used after the preemptive function call; and

b. providing instructions for loading the values assigned to the live variables from the local memory when they are required for subsequent execution of the main level function.

11. The method as recited in claim 9 where providing instructions for enabling the preemptive function to distinguish between its first and subsequent calls comprises providing instructions for initializing the thread's condition code, the condition code being a special field associated with the thread used for ascertaining the thread state, the initial value assigned to the field being indicative of the fact that the thread has not been preempted as yet due to the ensuing instance of the preemptive function.

12. The method as recited in claim 9 wherein providing instructions for facilitating the switching back of the thread comprises:

a. providing instructions for setting called function pointer of the thread to point to the preemptive function being called, the called function pointer being a special pointer associated with the thread, the pointer being subsequently used to point at the function that caused the thread to preempt;

b. providing instructions for saving parameters related to the function call in memory; and

c. providing instructions for setting up an executing stack for the preemptive function, the execution stack being needed for maintaining the activation records related to the preemptive function, the setting up of the stack being done in a manner such that an equivalent stack can be set up during switching back of the thread.

13. The method as recited in claim 8 wherein compiling the preemptive function call comprises providing instructions for using the same stack as used by the main level function, the stack being overwritten during the preemptive function call, overwriting of the stack having been facilitated by storing all pertinent information residing on the stack during compilation of main level function.

14. The method as recited in claim 7 wherein executing the compiled thread comprises:

a. setting up the thread for running;

b. running the thread by executing compiled code;

c. preempting the thread and switching to other threads when the thread requests preemption, the preemption being done without saving any context information, the context information being pertinent information associated with the thread; and

d. restarting the preempted thread when the thread is scheduled back for running, such restarting being done with a minimum amount of loading of context information.

15. The method as recited in claim 14 wherein the step of setting up the thread comprises:

a. setting up the execution stack for the activation records of the main level function and other functions within the thread, the thread stack capable of being overwritten during the execution of the thread; and

b. calling the main level function of the thread.

16. The method as recited in claim 14 wherein the step of restarting the thread comprises:

a. setting up the execution stack for execution of the preemptive function that caused the thread to preempt, the setting up being done in a manner such that

the subsequent execution of the thread may continue on the same stack, the setting up being done to make minimal use of memory; and

b. calling the preemptive function that caused the thread to preempt, the call being made from the beginning of the function, the calling being facilitated by the main level function having saved all information pertinent to the restarting of the preemptive function.

17. A multithreaded application processing system for executing a multithreaded application program, the program comprising a plurality of normal and floating threads, the normal threads having been written using standard thread methodology, the floating threads comprising a main level function and other sub-functions, the main level function being the basic execution level function making calls to other sub-functions, the sub-functions comprising preemptive and non-preemptive function calls, the sub-functions having been written using a function calling convention that ensures minimal thread switching overheads and reduced memory usage during application execution, the system comprising:

a. a compiler service compiling the application program, the compiler service using separate compilation methodologies for normal and floating threads;

b. a memory storing information related to the threads, the memory being allocated to the threads in a manner so as to minimize memory usage across thread switches;

c. a processor processing the plurality of threads, the processor accessing the memory for information pertaining to the plurality of threads; and

d. an operating system scheduling and managing execution of the plurality of threads, the operating system swapping floating threads without requiring to save reference information pertaining to the floating threads each time these threads are switched into and switched out of the processor in order to minimize the thread switching overheads.

18. The system as recited in claim 17 wherein the compiler service compiling the application program comprises:

a. a compiler compiling normal threads in accordance with the standard methodology for compiling threads; and

b. a floating thread compiler compiling floating threads in a manner such that no reference information needs to be stored on the processor when the floating thread is swapped out of the processor and subsequently swapped back into the processor.

19. The system as recited in claim 18 wherein the floating thread compiler comprises:

a. a main level function compiler compiling entry level function of the floating thread, the entry-level function being the basic execution level function that makes subsequent function calls;

b. a preemptive function compiler compiling functions that are preemptive in nature, the preemptive functions being those functions that may cause a thread to temporarily stall and yield control back to the operating system; and

c. a non-preemptive function compiler compiling function calls that are non-preemptive in nature, the non preemptive functions being those functions that cannot stall the thread at any stage of execution.

20. The system as recited in claim 17 wherein the memory allocated to normal and floating threads comprises:

a. a plurality of normal thread stacks, each of the normal thread stacks being associated with a normal thread, the stack being stored with context information pertaining to the thread, the context information being the entire information set relevant to the thread;

b. a plurality of floating thread stores, each of the thread stores being associated with a floating thread, the thread stores being stored with minimal context information that needs to be kept persistent across a floating thread switch; and

c. a floating thread stack associated with the processor, the floating thread stack being used by the floating thread being processed by the processor.

21. The system as recited in claim 17 wherein the multithreaded processing system works in a multiprocessor environment, each of the multiple processors having a separate floating thread stack associated with it.

22. The system as recited in claim 17 wherein the operating system managing the scheduling of threads comprises:

a. a scheduler ready queue holding threads that are ready for execution;

b. a normal thread preemption module providing preemptive services to normal threads, the preemptive services being specific activities that might preempt a thread; and

c. a floating thread preemption module providing preemptive services to floating threads.

23. The system as recited in claim 19 wherein the floating thread compiler further comprises:

a. means for saving reference information related to the floating thread in a thread store in response to a preemptive function call, the thread store being associated with the thread; and

b. means for setting up the floating thread stack in response to a preemptive function call, the setting up being done in a manner such that the no context information needs to be stored for the thread in case the thread is preempted during a preemptive function call.

24. A floating thread compiler compiling an application code, the application code comprising a plurality of floating threads, each floating thread comprising a main level function and other sub-functions, the main level function being the basic execution level function making calls to other sub-functions, the sub-functions having been written using a function calling convention, the convention ensuring minimal thread switching overheads and reduced memory usage during application execution by not requiring any reference information to be saved in the main memory when the thread is swapped out, the floating thread compiler comprising:

a. a main level function compiler compiling the main level function of the floating thread;

b. a non-preemptive function compiler compiling sub-functions that are non-preemptive in nature, the non preemptive functions being those functions that cannot stall the thread at any stage of execution, the non-preemptive functions being called from the main level function or any sub-function called by the main level function, the non-preemptive functions being capable of making calls to other non-preemptive functions only; and

c. a preemptive function compiler compiling sub-functions that are preemptive in nature, the preemptive functions being those functions that may cause a thread to temporarily stall and yield control back to the operating system, the preemptive functions being called from the main level function only, the preemptive functions being capable of making calls to other non-preemptive functions only.

25. The floating thread compiler as recited in claim 24 further comprising:

a. means for identifying the main level, preemptive and non-preemptive function blocks within the floating thread; and

b. means for activating the appropriate function compiler for compiling each of the functions.

26. A computer program product for minimizing thread switching overheads and memory usage while executing a multithreaded application program, the program having been written using floating threads, the floating threads following a function calling convention, the convention ensuring minimal switching overheads by not requiring any reference information to be saved in the main memory when the thread is swapped out, the computer program product comprising:

a computer readable medium comprising:

a. program instruction means for compiling floating threads in accordance with the function calling convention being followed by the floating threads; and

b. program instruction means for executing the compiled application program.

27. The computer program product as recited in claim 26 wherein the computer readable medium comprising program instruction means for compiling the floating threads further comprises:

a. program instruction means for compiling main level function, the main level function being the basic execution level function of the thread, the main level function making all subsequent calls to other functions;

b. program instruction means for compiling non-preemptive functions, non-preemptive functions being normal functions that do not stall the thread at any stage of execution, the non-preemptive functions being called from the main level function or any sub-function called by the main level function, the non-preemptive functions being capable of making calls to other non-preemptive functions only;

c. program instruction means for compiling preemptive functions, the preemptive functions being those functions that may cause a thread to temporarily stall and yield control to the operating system, the preemptive functions being called from the main level function only, the preemptive functions being capable of making calls to other non-preemptive functions; and

d. program instruction means for compiling other program constructs.